Indian Institute of Technology Patna
CS1101- Foundations of Programming

Lab-2: Introduction to Linux and programming

Date: 11-8-2025

(Create a directory named `Lab2` and navigate into it to perform your work)

Task1: Customizing Vim Editor Using .vimrc in Linux

Create a file .vimrc in your $HOME directory and type the following

$ touch .vimrc // To create an empty .vimrc file in home folder

Then type following command

To confirm whether it is created or not

$ ls -a // you should see .vimrc in the list

Edit .vimrc in vim

$ vi .vimrc then press i to enter insert mode

Then type

syntax on
set smartindent
set tabstop=4
set shiftwidth=4

press Esc and then type :wq

Command	Function
syntax on	Turns on syntax highlighting for the code
set smartindent	Provides automatic indentation while writing code
set tabstop=4	Sets the width of a Tab character to 4

	spaces
set shiftwidth=4	Sets the indentation shift width to 4 spaces

Now open one of your old c programs, hello.c on vim

And see the changes in it.

Can try these commands:

Command	Function
set number	Displays absolute line numbers in the left margin.
set showmatch	Highlights the matching bracket, brace, or parenthesis when the cursor is on one.
set hlsearch	Highlights all matches of the last search pattern.
set incsearch	Shows search matches as you type your search query (incremental search).

Task 2: Compiling C Programs and Using Input/Output Redirection in Linux

Create a file hello.c

```c
#include<stdio.h>

int main(){
        printf("Hello World !\n");
        printf("This is my first C program.\n");
        return 0;
}

```

Try the command:

To compile the code

$ gcc hello.c

To check weather default executable filed is built or not

$ ls

you should see a file named with a.out

To run the executable file and save the output in a text file

$ ./a.out > output.txt

To see the content of output text file

$ cat output.txt

Create a file namely myprog01.c and type the following

```
1
2 #include<stdio.h>
3 int main(){
4         int x;
5         printf("Enter an interger\n");
6         scanf("%d", &x);
7         printf("square of %d is %d\n", x, x*x);
8         return 0;
9 }
10
```

Try the following commands:

$ gcc myprog01.c

$ ls

To read standard input(keyboard)

$ ./a.out

Enter an integer

4

square of 4 is 16

Now Create a file namely input.txt and write a single integer in the file

$ echo 4 > input.txt

To see the content of the file

$ cat input.txt

Input redirection in Linux

$a.out < input.txt //The keyboard input is replaced with the contents of `input.txt`

$ ./a.out < input.txt > output.txt

The < input.txt takes whatever is inside `input.txt` and feeds it to your program as standard input (instead of you typing).

The > output.txt takes your program's standard output (which normally prints to the terminal) and saves it into `output.txt` (instead of showing it on screen).

$cat output.txt //To see the content of the output text file.

Task 3: Enter the following program using the `vi` editor, save it as `math_ops.c`, and compile it using different `gcc` options (`-Wall`, `-O2`, `-S`, `-E`, `-fsyntax-only`) to observe the output or generated files. Finally, run the program and note the output.

```c
#include <stdio.h>

int main() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    printf("Sum = %d\n", a + b);
    printf("Difference = %d\n", a - b);
    printf("Product = %d\n", a * b);
    printf("Quotient = %d\n", a / b);
    printf("Remainder = %d\n", a % b);

    return 0;
}
```

Command	Description
gcc math_op.c -o math_op	Basic compilation
gcc -Wall math_op.c -o math_op	Show all warnings
gcc -E math_op.c -o math_op.i	Stop after preprocessing, output to .i file
gcc -S math_op.c	Stop after compilation, output assembly .s file
gcc -c math_op.c	Compile to object .o file, no linking
gcc -O2 math_op.c -o math_op	Compile with optimization level 2
gcc -fsyntax-only math_op.c	Check syntax only, no output file

What difference you find while using

gcc -O0 -S math_ops.c -o O0.s

gcc -O2 -S math_ops.c -o O2.s

Compare file sizes in each case and tabulate

Command	File Size
gcc math_op.c -o math_op	16064 Bytes
gcc -Wall math_op.c -o math_op	
gcc -E math_op.c -o math_op.i	
gcc -S math_op.c	
gcc -c math_op.c	
gcc -O2 math_op.c -o math_op	
gcc -O0 -S math_ops.c -o O0.s	
gcc -O2 -S math_ops.c -o O2.s	

Task: Create the C files `gen_data.c` and `sum_from_file.c` `(use vi editor)`, then compile and execute them using the shell script `run.sh`.

```c
// "gen_data.c"
#include <stdio.h>
int main() {
    int a, b, c;
   // printf("Enter first integer: ");
    scanf("%d", &a);
    //printf("Enter second integer: ");
    scanf("%d", &b);
    //printf("Enter third integer: ");
    scanf("%d", &c);
    printf("%d\n", a);
    printf("%d\n", b);
    printf("%d\n", c);
    return 0;
}
```

```c
// "sum_from_file.c"
#include <stdio.h>
int main() {
    int a,b,c;
    //printf("Reading integers...\n");
    scanf("%d", &a);
    scanf("%d", &b);
    scanf("%d", &c);
    printf("Sum = %d\n", a+b+c);
    return 0;
}
```

```
################ run.sh ##############

#!/bin/bash
set -e

# Compile both programs
gcc gen_data.c -o gen_data
gcc sum_from_file.c -o sum_from_file

# Run first program, redirect output to file
./gen_data > numbers.txt

# Run second program, taking input from file
./sum_from_file < numbers.txt
```

Task4: **The OS checks a program's return value (also called exit status) to decide if it "succeeded" or "failed." By convention: 0 → Success Non-zero → Error (different numbers can mean different errors). We can show this with a simple Hello World program that uses different `return` values and then check `$?` in the shell.**

```c
/////hello_exit.c //////////

#include <stdio.h>

int main() {
    printf("Hello, World!\n");

    int choice;
    printf("Enter return code (0 for success, non-zero for error): ");
    scanf("%d", &choice);

    return choice; // send this back to the OS
}
```

>gcc hello_exit.c -o hello_exit

>./hello_exit

>echo $?

$ ./hello_exit

Hello, World!

Enter return code (0 for success, non-zero for error): -1

$ echo $?

What do you see?

Simulating OS Behavior

We can use shell scripts to react to the exit code.

```
############## check_exit.sh #################
#!/bin/bash
gcc hello_exit.c -o hello_exit
./hello_exit
status=$?
if [ $status -eq 0 ]; then
    echo "OS sees: Program succeeded."
else
    echo "OS sees: Program failed with code $status."
fi
##################check_exit.sh #################
```

Task5: Temporarily adding your current directory to PATH

(Remember :By default, the current directory (`.`) is not in `$PATH` for security reasons, so `a.out` won't be found unless you type `./a.out`)

In command promptType: export PATH="$PATH:."

Then try a.out What do you see?

……

You can also add path to to your .bashrc file

(.bashrc is a hidden shell configuration file in your home directory that is executed every time you start a new interactive non-login Bash session.

It is used to set environment variables, customize the command prompt, define aliases, and run startup commands, allowing you to personalize your shell environment.)

```
vi ~/.bashrc

# is the same as

vi $HOME/.bashrc

# is the same as

vi /home/student/.bashrc
```

Try adding current directory to your .bashrc and then try a.out

Task 6: Write a simple C program and execute it from the current directory

For eg.

```c
#include <stdio.h>

int main() {
    float celsius, fahrenheit;
    printf("Enter temperature in Celsius: ");
    scanf("%f", &celsius);
    fahrenheit = (celsius * 9 / 5) + 32;
    printf("%.2f Celsius = %.2f Fahrenheit\n", celsius, fahrenheit);
    return 0;
}
```

In Record:

Document the outputs of the lab tasks and summarize the key learnings from today's session